

Muon Identification Software Overview

Currently Implemented Muon ID Software

In mMuiMuonId.cc:

```
if ((last_plane>=1) && (last_plane <=4)) { // first plane is 0.  
    if ( (p.mag() < dMuiMuonIdProfilePar[0].pmax[last_plane]) p.mag is momentum at station 3  
        && (fabs(vx) < dMuiMuonIdProfilePar[0].xVertexCut)  
        && (fabs(vy) < dMuiMuonIdProfilePar[0].yVertexCut) ) {  
        dMuoTrackRoadRel[dMuoTrackRoadRelId].MuonProbability=1.0;  
    }  
}
```

In muifuncs.C:

```
dMuiMuonIdProfilePar->set_pmax(0, 0, 0.0);  
dMuiMuonIdProfilePar->set_pmax(1, 0, 1.0);  
dMuiMuonIdProfilePar->set_pmax(2, 0, 1.3);  
dMuiMuonIdProfilePar->set_pmax(3, 0, 1.7);  
dMuiMuonIdProfilePar->set_pmax(4, 0, 200.0);  
dMuiMuonIdProfilePar->set_xVertexCut(0, 80.0);  
dMuiMuonIdProfilePar->set_yVertexCut(0, 80.0);
```

Methods Which Have Been Investigated

- Discriminate analysis (Yajun)
- Look Up Table (Jason)

For both methods, bulk of development time should be in choosing best variables and “training” the method.

No code for either in CVS. (This is Ok. No Legacy code to shoehorn into any new frame work)

Look Up Table Basic Idea

- Fill Mu[momentum][lastGap][]... (muons) and Pi[momentum][lastGap][]... (pions) from simulation.
(could be broken down into MuPlus[], MuMinus[],...)
- Muon Probability is then
 $\text{Mu}[p][g][] / (\text{Mu}[p][g][] + K * \text{Pi}[p][g][])$
K is very important!
(Well hopefully not, but then we wouldn't need this software)

Some Possible LUT variables

- Momentum (3rd station or vertex?), Last Gap, maxhits (probably not), maxClusterWidth, vertex.
- Other suggestions?

Judging the LUT

- Works well on simulated data.
- Fairly robust to changes in a “reasonable” K value.
- A good figure of merit might be:
% of bins with $a < \text{Mu prob} < b$ ($a=0.1, b=0.9?$)
How does this change when you add a variable to the table, change K, rebin ...?

A Few LUT Filling Questions

- Is single particle simulation sufficient?
- Is it OK to leave dealing with detector efficiency (and other effects) to the roadfinder, or should this be included in the “training” somehow?
- How do we get a “reasonable” K?

Commonalities for any Identifier Software

- Should store parameters in the database.
For me this means at least LUT(s), maybe K.
- Should inherit from a base class that provides a simple common interface.
- Should work at DST and μ DST level.
- Should not reject events/roads.
- Should be method be switchable without recompile?